End-to-End Language-Action Model for Humanoid Whole Body Control

Supplementary Material

A. Language-Action Dataset

A.1. Humanoid Hardware

This work uses the **Unitree G1**, a lightweight humanoid robot equipped with 29 actuated degrees of freedom (DoF): 7 DoF for each arm, 6 DoF for each leg, and 3 Dof at the waist. The robot uses a standard low-level joint-space Proportional–Derivative (PD) controller, which computes motor torques τ according to:

$$\tau = K_p(q^{\text{target}} - q) - K_d\dot{q}, \tag{8}$$

where $q \in \mathbb{R}^{29}$ and $\dot{q} \in \mathbb{R}^{29}$ denote the measured joint angles and velocities, and $q^{\text{target}} \in \mathbb{R}^{29}$ is the desired target joint position commanded by a policy controller. The PD controller runs at 200 Hz, and the control policy runs at 50 Hz.

A.2. Whole Body Control Tracking Policy

We define the reference motion m_t^{ref} for tracking as:

$$m_t^{\text{ref}} = \left[x_t^{\text{ref}}, r_t^{\text{ref}}, q_t^{\text{ref}}, \dot{q}_t^{\text{ref}}, p_t^{\text{ref}}\right], \tag{9}$$

where $x_t^{\text{ref}} \in \mathbb{R}^3$ denotes the root translation, $r_t^{\text{ref}} \in \text{SO}(3)$ denotes the root rotation, $q_t^{\text{ref}}, \dot{q}_t^{\text{ref}} \in \mathbb{R}^{29}$ denotes the joint positions and velocities, and $p_t^{\text{ref}} \in \mathbb{R}^{4\times3}$ denotes the keypoint positions of the wrists and ankles in the local body frame.

The tracking policy π_{track} takes as input the proprioceptive state s_t and motion-tracking target m_t^{ref}, which are define in Equation (1) and Equation (9), respectively. The policy outputs a_t, which is linearly projected to compute the target joint positions q_t^{target} for PD controller:

$$q_t^{\text{target}} = a_t \cdot \alpha + \bar{q}, \tag{10}$$

where α denotes the scale parameters and $\bar{q}$ represents a constant nominal joint configuration, following the convention in [31].

To train the whole body control tracking policy, we use PPO [48] as our RL algorithm and IsaacLab [40] as the training environment. We implement a Mixture-of-Expert (MoE) policy to enhance its capability. The training hyperparameters, domain randomization, and reward terms are listed in Table 6, Table 7, and Table 8, respectively. Following [56], we apply PHC [37] to filter the AMASS dataset [38], resulting in 8,179 high-quality motion sequences for whole body control training.

Table 6. Hyperparameters for tracking policy training.

Term	Value
Num parallel envs	4096
Num steps per env	24
Learning epochs	5
Num mini_batches	4
Discount γ	0.99
GAE λ	0.95
PPO clip ratio	0.2
Value loss coefficient	1.0
Entropy coefficient	0.005
Initial learning rate	1e-3
Adaptive LR range	1e-5, 1e-2
Desired KL	0.01
Max grad norm	1.0
Optimizer	Adam
MLP layers	[512, 256, 128]
Num experts	12

Table 7. Domain randomization for tracking policy training.

Domain Randomization	Sampling Distribution
Static friction	$\mathcal{U}(0.3, 1.6)$
Dynamic friction	$\mathcal{U}(0.3, 1.2)$
Restitution	$\mathcal{U}(0, 0.5)$
Default joint position	$\mathcal{U}(-0.01, 0.01)$
Center of mass	$\mathcal{U}(-0.05, 0.05)$
Random push interval	$\mathcal{U}(1.0, 3.0)$
Random push linear velocity	$v_x : \mathcal{U}(-0.5, 0.5)$
	$v_y : \mathcal{U}(-0.5, 0.5)$
	$v_z : \mathcal{U}(-0.2, 0.2)$
Random push angular velocity	$\omega_r : \mathcal{U}(-0.52, 0.52)$
	$\omega_p : \mathcal{U}(-0.52, 0.52)$
	$\omega_y : \mathcal{U}(-0.78, 0.78)$

A.3. Dataset Composition

We construct our language–action dataset by tracking human motion sequences using the trained whole body control policy. We use a subset of the policy training data whose motion sequences are annotated with language descriptions

Table 8. Rewards for tracking policy training.

Term	Expression	Weight
Task Reward		
Root position	$\exp(-\|x_t - x_t^{\text{ref}}\|/0.09)$	0.5
Root rotation	$\exp(-\|r_t - r_t^{\text{ref}}\|/0.16)$	0.5
DoF position	$\exp(-\|q_t - q_t^{\text{ref}}\|/0.16)$	1.0
DoF velocity	$\exp(-\|\dot{q}_t - \dot{q}_t^{\text{ref}}\|/1.00)$	1.0
Keypoint position	$\exp(-\|p_t - p_t^{\text{ref}}\|/0.09)$	1.0
Keypoint linear velocity	$\exp(-\|\dot{p}_t - \dot{p}_t^{\text{ref}}\|/1.00)$	1.0
Penalty		
Action rate	$-\|a_t - a_{t-1}\|$	0.1
DoF limit	$-\mathbb{I}(q_t \notin [q_{\min}, q_{\max}])$	10.0
Undesired contacts	$-\sum_{c \notin \{\text{ankles,wrists}\}} \mathbb{I}(\|\mathbf{F}_c\| > 1.0N)$	0.1

in HumanML3D [13], along with their mirrored version, resulting in 12,422 motion clips, each paired with three to four textual descriptions. For each motion sequence, we roll out the trained policy 20 times under domain randomization to obtain diverse robot trajectories corresponding to the same target motion. The domain randomization applied during data collection includes those for policy traning, as listed in Table 7, along with action noise sampled form $\mathcal{U}(-0.02, 0.02)$ and added to the target joint position q_t^{target}. To ensure dataset quality, we retain only successful trajectories when constructing the final language–action dataset $\mathcal{D}_{\text{robot}}$.

B. Language-Action Model

B.1. Model Architecture

Our model builds upon the architecture of SmolVLA [50], with several modifications tailored for text-based humanoid whole-body control.

We adopt the architecture of SmolVLA as our backbone, but we do not initialize the model with any pretrained VLM or VLA weights. Instead, we train it from scratch on our language–action dataset. In addition, we omit the vision encoder and retain only the language-model component of the architecture for text conditioning and state history encoding. For the 600M model (our default model in experiments), we use `SmolVLM-Base` with num_vlm_layers set to 6. For the 200M and 60M variants, we adopt `SmolVLM2-500M-Video-Instruct` and `SmolVLM2-256M-Video-Instruct`, respectively, each configured with num_vlm_layers set to 12. We do not adopt layer skipping for the action expert, so the action expert shares the same number of layers as the backbone.

We first encode the language command using the CLIP text encoder [46] to obtain semantic token embeddings. Both the text embeddings and the proprioceptive state inputs are then projected into the hidden dimension of the language–state encoder through separate linear layers. To distinguish between modalities, we add learnable type embeddings to the projected language and state tokens. Since the state history is provided at multiple temporal granularities, as defined in Equation (5), we additionally incorporate learnable time embeddings for the state tokens, enabling the model to capture their temporal relationships. The action expert employs one linear projector to embed the sampled action noise into its hidden dimension, and another to decode the expert's hidden states back into the action space.

The state history consists of short-term states $\mathbf{s}_t^{\text{short\_term}}$ sampled at 50 Hz and long-term states $\mathbf{s}_t^{\text{long\_term}}$ sampled at 4 Hz, defined as:

$$\mathbf{s}_t^{\text{short\_term}} = [s_{t-0.18}, s_{t-0.16}, \ldots, s_{t-0.02}, s_t], \quad (11)$$

$$\mathbf{s}_t^{\text{long\_term}} = [s_{t-10}, s_{t-9.75}, ..., s_{t-0.5}, s_{t-0.25}]. \quad (12)$$

Such a multi-scale observation preserves high-frequency control feedback while exposing long-horizon temporal structure to the model.

The done prediction head is a two-layer MLP with a hidden dimension same as that of the language–state encoder. It takes as input the the last hidden state of the final state token (*i.e.*, s_t) and outputs a scalar indicating the probability that the task will be completed within the next H steps, *i.e.*, the length of the action chunk.

B.2. Training Details

We train our language–action model using the AdamW optimizer with a batch size of 1024 distributed across 4 NVIDIA A800-SXM4-80GB GPUs. The initial learning rate is set to 1e-4, with a cosine decay schedule applied over 100k gradient update steps to 2.5e-6. During training, we randomly mask language inputs with a probability of 0.1 to

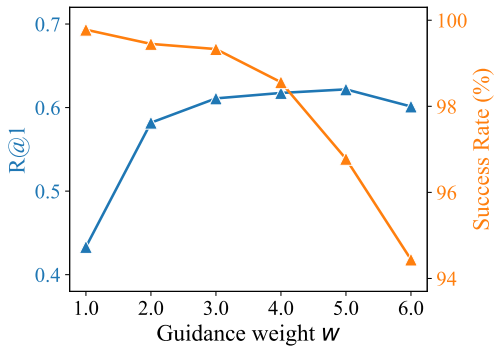

Figure 7. Effect of classifier-free guidance weight w.

facilitate classifier-free guidance. The model is trained for 100k gradient update steps, which takes approximately 24 hours.

The training objective consists of two components: a primary flow matching loss and a done prediction loss. The flow matching loss L_{fm} is defined in Equation (3). The done prediction loss L_{done} is a binary cross-entropy loss that encourages the model to estimate whether the task will terminate within the next action chunk. The overall loss is:

$$L = L_{\text{fm}} + \lambda L_{\text{done}}, \tag{13}$$

where $\lambda = 0.01$. To prevent interference with humanoid control learning, gradients from the done prediction head are not propagated back into the language–state encoder. Thus, L_{done} updates only the parameters of the done prediction head.

B.3. Classifier-Free Guidance

During inference, we apply classifier-free guidance to enhance the alignment between generated actions and language commands. Specifically, we perform two forward passes through the model at each denoising step: one with the actual language input and another with the language input masked out. The final predicted action noise is computed as a weighted combination of the two outputs:

$$v_{\text{cond}} = v_\theta(\mathbf{A}_t^\tau, \tau, [l, \mathbf{s}_t^{\text{hist}}]) \tag{14}$$

$$v_{\text{uncond}} = v_\theta(\mathbf{A}_t^\tau, \tau, [\varnothing, \mathbf{s}_t^{\text{hist}}]) \tag{15}$$

$$v_{\text{cfg}} = v_{\text{uncond}} + w(v_{\text{cond}} - v_{\text{uncond}}) \tag{16}$$

$$\mathbf{A}_t^{\tau-\Delta t} = \mathbf{A}_t^\tau - v_{\text{cfg}} \cdot \Delta t. \tag{17}$$

Here, $w > 1$ is the guidance weight that amplifies the influence of the language command on the generated actions. We conduct an experiment on the guidance weight as shown in Figure 7. The R@1 improves as we increase w from 1.0 to 5.0, indicating better alignment between the generated actions and language commands. However, the success rate decreases with larger w, suggesting that overly aggressive guidance may lead to less stable control. This indicates a trade-off between physical fidelity and language alignment when tuning the guidance weight for text-based whole body control. We choose $w = 2.0$ as a balanced setting for our main experiments.

Table 9. Ablation study results for residual post-training.

Method	R@1 ↑	MMD ↓	Success ↑
Base + π_Δ	**0.392**	**6.100**	99.11
Base + π_Δ w/o r^{DoF}	0.086	25.358	**99.89**
Base + π_Δ w/o r^{Done}	0.255	9.779	99.78

B.4. Residual Post-Training

To train π_Δ, we freeze the parameters of the language–action model and optimize the residual head with PPO [48] in IsaacLab [40]. We apply domain randomization over physical parameters and external perturbations to improve the robustness of our method, same as those used during the tracking policy training, as listed in Table 7. The residual action head π_Δ is implemented as an MLP with three hidden layers of sizes [512, 256, 128], and is trained with the same PPO hyperparameters listed in Table 6, except that the number of parallel environments is reduced to 1024 due to GPU memory limits. Before being added to the original action predicted by the language–action model, the residual action Δa_t is scaled by a factor of 0.02.

The reward functions as summarized in Table 10. The DoF position tracking term encourages the residual action to drive the future joint positions toward the original predictions $\hat{q}$ under domain randomization. The active termination means that the language-action model predicts the task is done, which encourages the residual policy to complete the task as intended. The residual action norm term penalizes large residual corrections to prevent the policy from deviating excessively from the original action.

To evaluate the effectiveness of the two task reward terms, *i.e.*, DoF (joint) position tracking r^{DoF} and active termination r^{Done}, we conduct an ablation study during residual post-training. As shown in Table 9, removing active termination results in a significant reward hacking, where the residual action head learns to output corrections that make the humanoid robot control overly conservative, leading to a high success rate but very poor language following. In practice, we observe that the robot tends to maintain a stable standing pose without performing any movements, simply to avoid the large penalty from falling. Similarly, removing the DoF position tracking term also leads to conservative behavior, albeit to a lesser extent than the removal of active termination.

Table 10. Rewards for residual post-training.

Term	Expression	Weight
Task Reward		
DoF position	$\exp(-\|q_t - \hat{q}_t\|/0.16)$	2.0
Active termination	1.0 if LA model predicts done	100.0
Penalty		
Residual action norm	$\exp(-\|\Delta a_t\|/0.09)$	2.0
Action rate	$-\|a_t^{\text{final}} - a_{t-1}^{\text{final}}\|$	0.2
DoF limit	$-\mathbb{I}(q_t \notin [q_{\min}, q_{\max}])$	10.0
Undesired contacts	$-\sum_{c \notin \{\text{ankles,wrists}\}} \mathbb{I}(\|\mathbf{F}_c\| > 1.0N)$	0.1
Termination	-1.0 if fall down	100.0

C. Experiment Details

C.1. Baselines

MDM/T2M-GPT + Retarget. We use the official checkpoints of MDM [53] and T2M-GPT [62] to generate human motion sequences from our test language commands. The generated motions are then retargeted to humanoid robot kinematics using the same retargeting pipeline as in our tracking policy training. The resulting robot motions are executed using our tracking policy.

UH-1. We reproduce UH-1 [39] based on the T2M-GPT codebase, but replacing human motion with humanoid robot motion obtained by the same retargeting pipeline as in our tracking policy training. The generated robot motion is subsequently executed via our tracking policy.

LangWBC. We implement LangWBC [49] within our whole body control policy training framework and distill the CVAE-based policy using our tracking policy as the teacher model.

We use the same tracking policy for motion execution (MDM/T2M-GPT + Retarget and UH-1), policy distillation (LangWBC), and BC data collection (ours) to ensure a fair comparison, where variations arising from the underlying whole-body controller are effectively removed.

C.2. Evaluation Metrics

We train a TMR [45] on our language-action dataset to learn the alignment between language commands and real robot trajectories. Specifically, each trajectory is downsampled to 10Hz, and the input for each frame consists of (i) root linear velocity and root angular velocity, (ii) joint positions and joint velocities, and (iii) the local translations and 6D rotations of 29 rigid bodies.

Based on the trained TMR and collected evaluation trajectories $\mathcal{D}_{\text{robot}}^{\text{eval}} = \{(\boldsymbol{\tau}_i, l_i)\}_{i=1}^{N}$, we first compute a similarity matrix $S \in \mathbb{R}^{N \times N}$, where each element is defined as:

$$S_{i,j} = \frac{f_{\text{TMR}}^{\text{t}}(l_i)^\top f_{\text{TMR}}^{\text{m}}(\boldsymbol{\tau}_j)}{\|f_{\text{TMR}}^{\text{t}}(l_i)\| \|f_{\text{TMR}}^{\text{m}}(\boldsymbol{\tau}_j)\|}. \tag{18}$$

Here, $f_{\text{TMR}}^{\text{t}}(\cdot)$ denotes the text encoder of TMR, and $f_{\text{TMR}}^{\text{m}}(\cdot)$ represents the motion encoder of TMR. Using the similarity matrix S, we compute the following evaluation metrics:

Multi-Modal Distance. This metric measures the average distance between the encoded language command and the encoded robot trajectory in the TMR embedding space:

$$\text{MM-Dist} = \frac{1}{N} \sum_{i=1}^{N} (1 - S_{i,i}) \,. \tag{19}$$

R-Precision at K. This metric evaluates the retrieval accuracy of matching language commands to robot trajectories. Following [13], for each language command l_i, we rank a batch of 32 robot trajectories, including the paired one, based on their similarity scores $S_{i,j}$. R-Precision at K (R@K) is defined as the proportion of language commands for which the correct trajectory $\boldsymbol{\tau}_i$ is among the top K retrieved trajectories.

Diversity. This metric measures the variability of generated robot trajectories in the TMR embedding space. Following [13], it is computed as the average pairwise distance between two randomly sampled trajectory subset from the evaluation set. Formally, given two subsets $\mathcal{A}, \mathcal{B} \subset \mathcal{D}_{\text{robot}}^{\text{eval}}$, each containing $N_s = 300$ trajectories, the diversity is defined as:

$$\text{Diversity} = \frac{1}{N_S} \sum_{i=1}^{N_s} (1 - S_{a_i,b_i}), \tag{20}$$

where a_i and b_i are the indices of the i-th trajectory in subsets $\mathcal{A}$ and $\mathcal{B}$, respectively. We report the average diversity over 10 random pairs of subsets.

Maximum Mean Discrepancy. This metric measures the distributional difference in the TMR embedding space between the generated robot trajectories $\{\boldsymbol{\tau}_i\}_{i=1}^{N}$ and the ground-truth robot trajectories $\{\boldsymbol{\tau}_i^{\mathrm{GT}}\}_{i=1}^{M}$ collected by our tracking policy. Denote the TMR encoded trajectories as $e_i = f_{\mathrm{TMR}}^{\mathrm{m}}(\boldsymbol{\tau}_i)$ and $e_i^{\mathrm{GT}} = f_{\mathrm{TMR}}^{\mathrm{m}}(\boldsymbol{\tau}_i^{\mathrm{GT}})$. Following [22], we compute MMD as:

$$\begin{aligned}\mathrm{MMD} = &\frac{1}{N(N-1)}\sum_{i=1}^{N}\sum_{j\neq i}^{N} k(e_i, e_j) \\ &+ \frac{1}{M(M-1)}\sum_{i=1}^{M}\sum_{j\neq i}^{M} k(e_i^{\mathrm{GT}}, e_j^{\mathrm{GT}}) \\ &- \frac{2}{NM}\sum_{i=1}^{N}\sum_{j=1}^{M} k(e_i, e_j^{\mathrm{GT}}),\end{aligned} \tag{21}$$

where $k(x, y) = \exp(-\|x-y\|^2/2\sigma^2)$ is the Gaussian RBF kernel with bandwidth $\sigma = 10$. We scale up the MMD value by a factor of 100 for better readability.

C.3. Waypoint navigation

To construct a navigation-oriented subset of our language–action dataset, we design a set of 200 command templates that specify target positions in a 2D plane. The templates are structured to describe a person walking to a specific coordinate, with variations in phrasing to enhance linguistic diversity. These templates are generated using ChatGPT. Examples of these templates are provided below:

Template examples:
"A person walks to position (*x* unit, *y* unit)."
"Someone moves toward x = *x* unit, y = *y* unit."
"A man goes to location (*x* unit, *y* unit)."
"A woman walks toward (*x* unit, *y* unit)."
"A human steps to (*x* unit, *y* unit)."
"An individual proceeds to coordinate (*x* unit, *y* unit)."
"A pedestrian walks toward x = *x* unit, y = *y* unit."
"A man strides to (*x* unit, *y* unit)."
"A woman heads toward (*x* unit, *y* unit)."

Based on the displacements from the starting frame to the ending frame of approximately 2,000 walking-type motions in the AMASS dataset [38], we generate movement commands by filling in the target coordinates in the templates, using 20 cm as the discretization unit. For instance, if a motion sequence involves a displacement of (1.2 m, 0.8 m), we would create a command like "*A person walks to position (6 unit, 4 unit)*". We randomly apply three templates to each trajectory and finetune our language-action model on this dataset.

To quantitatively evaluate waypoint tracking accuracy, we randomly sample target positions within the range of 4m to 6m along the x-axis and -2m to 2m along the y-axis, and conduct a two-round navigation task. A navigation round is considered successful if the robot reaches within 0.5m of the target position and will skip next round. For the second iteration, we transform and discretize the target relative to the robot's frame at the end of the first round. For each navigation round, we sample one of the templates and fill in the discretized target coordinates to form the language input.

Table 11. Inference time of our model on a NVIDIA GeForce RTX 4090 GPU.

Model Part	**Inference Time**
LA model with 10 flow match pass	26ms
LA model with 5 flow match pass	16ms
Communication overhead	2ms

C.4. Real World Deployment

For real-world deployment, we use the same Unitree G1 humanoid robot as in simulation. The language–action model runs on an NVIDIA GeForce RTX 4090 GPU, which communicates with the robot through a wired connection. Proprioceptive states are streamed from the robot to the GPU at 50 Hz, where the model performs action generation via flow matching. The predicted actions are then returned to the onboard PD controller, operating at 200 Hz.

Unlike in simulation, where physics can be paused while waiting for the next inference step, the real robot continues to execute the last action during model computation. To accommodate this, we adopt an asynchronous inference pipeline [50], which decouples action generation from the control loop. The policy produces an action chunk of length K_1 every K_2 steps, with $K_1 > K_2$. Because the action expert is a causal transformer, it can generate shorter action chunks during evaluation than the full training horizon H used for training. While a new chunk is being generated, the robot continues executing the unconsumed actions from the previous chunk, ensuring uninterrupted control.

After a new action chunk becomes available, the system discards the first K_3 actions, corresponding to the steps elapsed during model inference, and replaces the previous action chunk with the remaining actions. Here, $K_3 = \lceil T_{\mathrm{infer}}/0.02 \rceil$, with T_{infer} denoting the model's inference latency and 0.02 s corresponding to the 50 Hz control cycle. In our experiments, we use $K_1 = 6$ and $K_2 = 4$, matching the rollout step used in our simulation experiments.

As shown in Table 11, a single inference of the language–action model takes 26 ms using 10 flow matching steps and 16 ms with 5 steps. Since the latter provides comparable performance in real-world tests, we adopt the 5-step flow matching for most hardware experiments to reduce latency. The communication overhead between the robot and

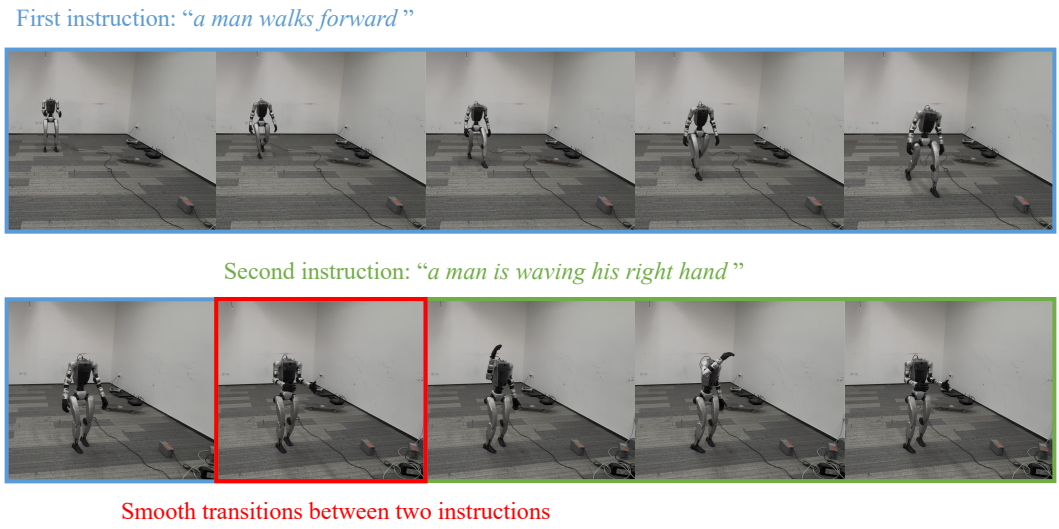

Figure 8. Autonomous sequential execution of two instructions: "*a man walks forward*" and "*a man is waving his right hand*". Our system produces smooth, continuous transitions between two instructions.

the GPU is approximately 2 ms. Overall, the asynchronous inference pipeline effectively compensates for model latency and enables smooth and robust whole-body control on real humanoid hardware. Besides Figure 6, we provides real world deployment videos in the supplementary materials to demonstrate that our model and inference pipeline can reliably execute diverse language instructions.

Autonomous Sequential Execution. Building on this capability, our system also supports autonomous sequential execution of multiple language commands. During dataset construction, each motion sequence is augmented with smooth transitions to and from a default standing pose. As a result, the language–action model learns a consistent behavioral pattern: it initiates each command from the default pose and returns to the same pose after completion[2]. This default pose anchoring enables the robot to execute multiple commands consecutively without manual reset: after returning to the default standing pose, the model's done prediction is used to autonomously terminate the current command and proceed to the next one, which again begins from the same default pose. Consequently, the system produces smooth, continuous transitions between behaviors and supports long-horizon control, as shown in Figure 8. **Full video demonstrations are provided in the supplementary materials.**

[2]To ensure fair comparison, we apply the same preprocessing to all baseline methods in previous experiments.